

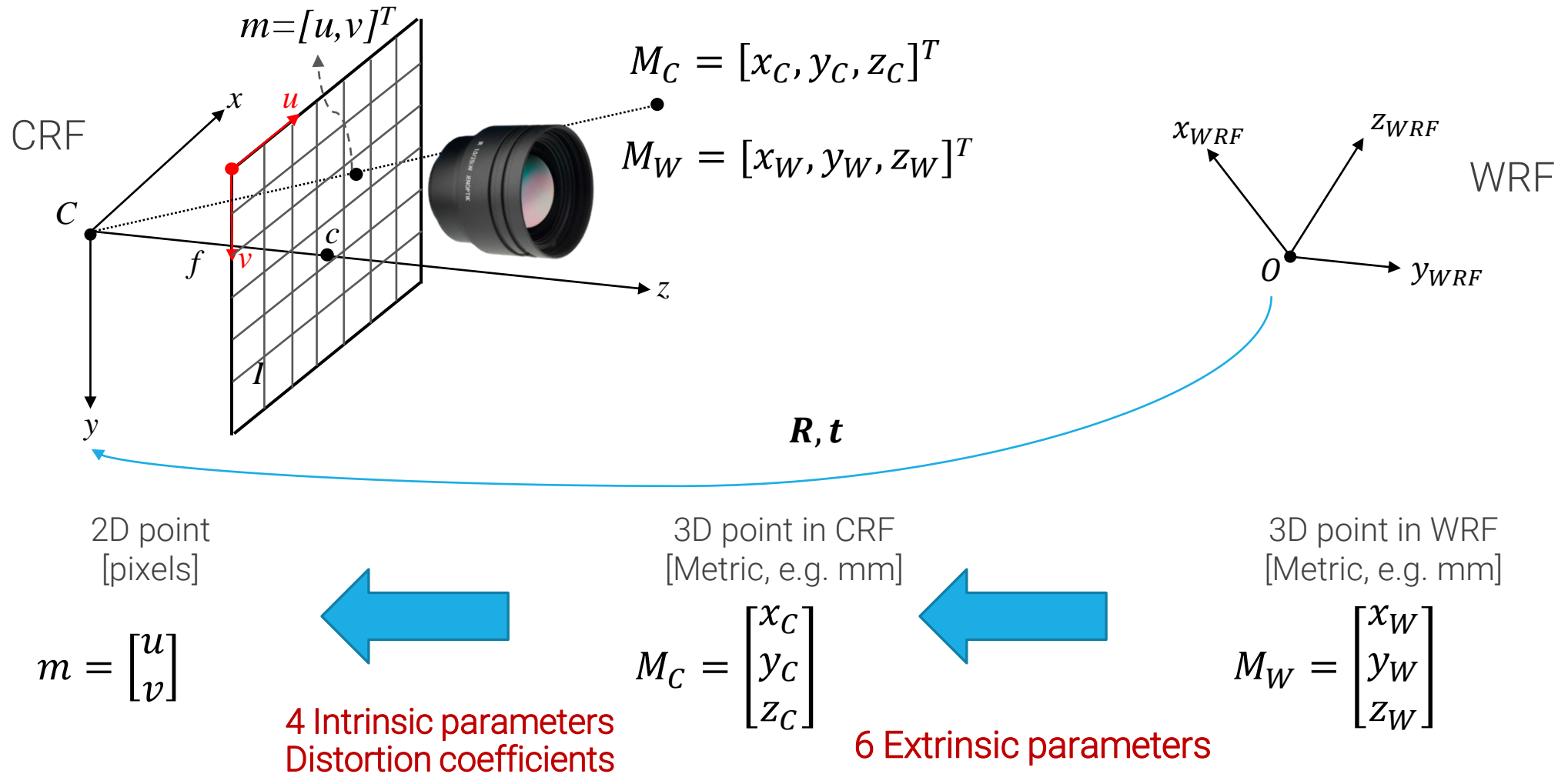
Lecture 2

Camera Calibration

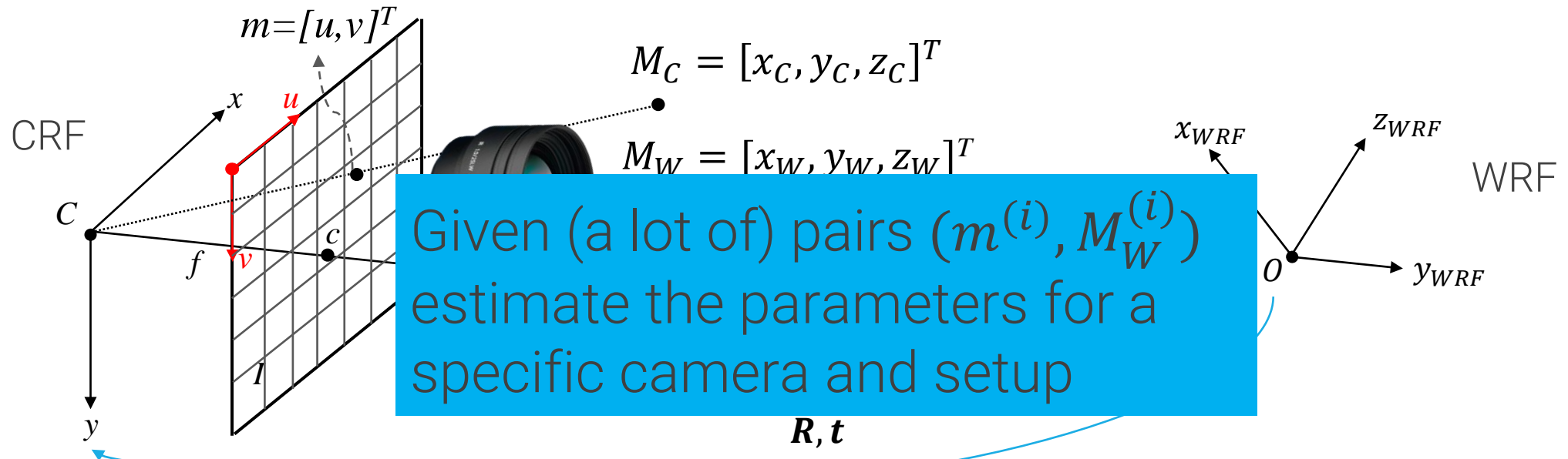
IMAGE PROCESSING AND COMPUTER VISION – PART 2

SAMUELE SALTI

Complete camera model



Camera calibration



Known

2D point
[pixels]

$$m = \begin{bmatrix} u \\ v \end{bmatrix}$$

4 Intrinsic parameters
Distortion coefficients

3D point in CRF
[Metric, e.g. mm]

$$M_C = \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix}$$

Unknown

6 Extrinsic parameters

3D point in WRF
[Metric, e.g. mm]

$$M_W = \begin{bmatrix} x_W \\ y_W \\ z_W \end{bmatrix}$$

Known

Calibration patterns

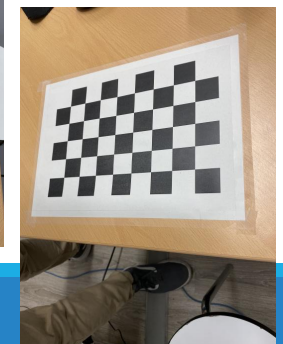
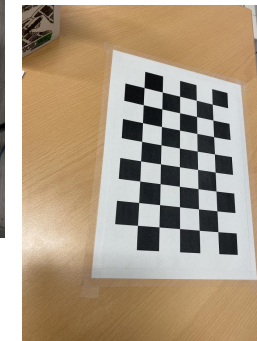
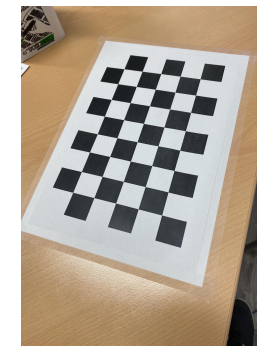
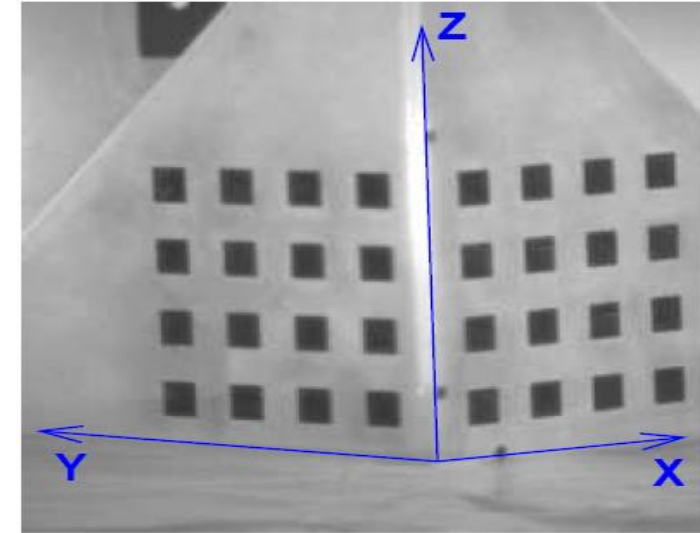
Camera calibration approaches can be split into two main categories:

- Those relying on **a single image of a 3D calibration object**
 - featuring several (at least 2) planes containing a known pattern
- Those relying on **several (at least 3) different images of one given planar pattern**

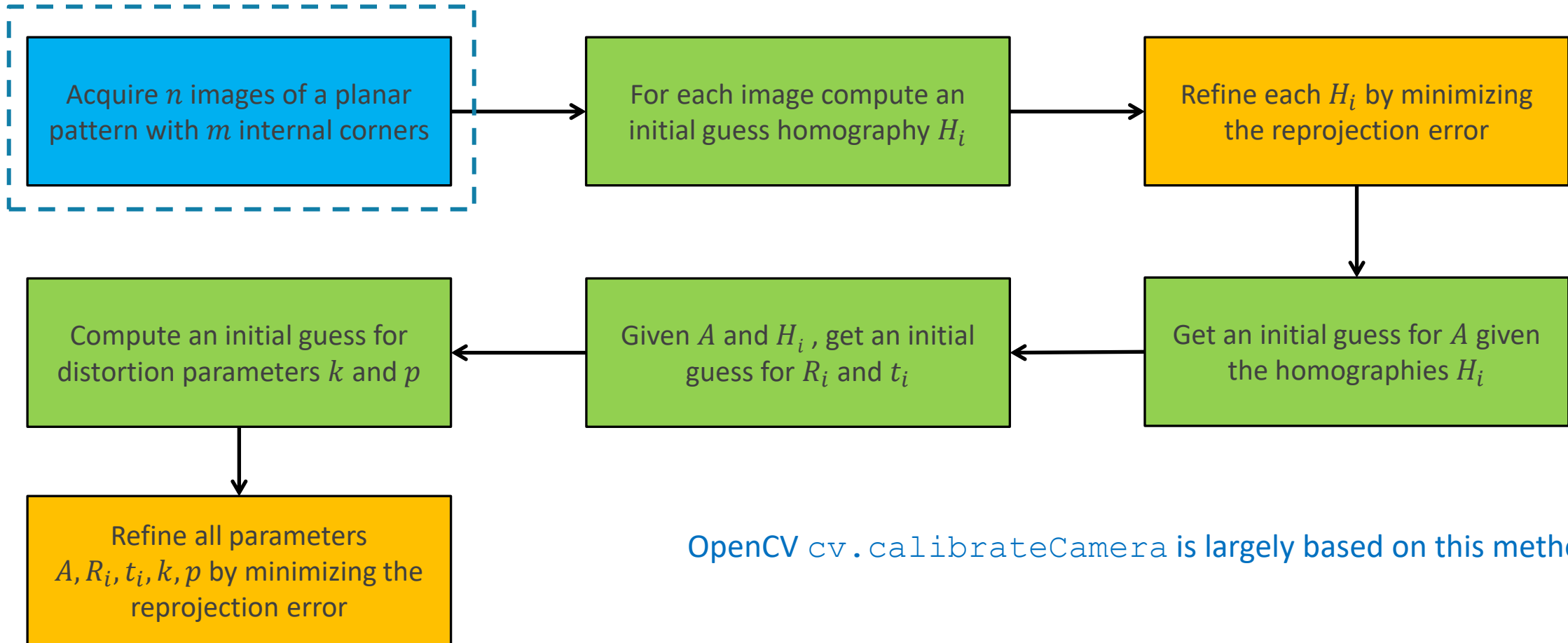
In practice, it is difficult to build accurate targets containing multiple planes, while an accurate planar target can be attained rather easily

Implementing a camera calibration software requires a significant effort

- the main Computer Vision toolboxes include specific functions (OpenCV, Matlab CC Toolbox)



Zhang's Method



OpenCV `cv.calibrateCamera` is largely based on this method

Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Trans. on PAMI, 2000.
<https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr98-71.pdf>

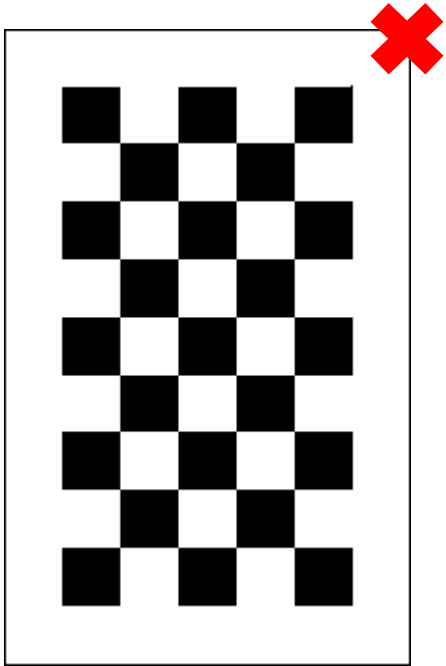
Calibration pattern

→ THE BIGGER THE BETTER

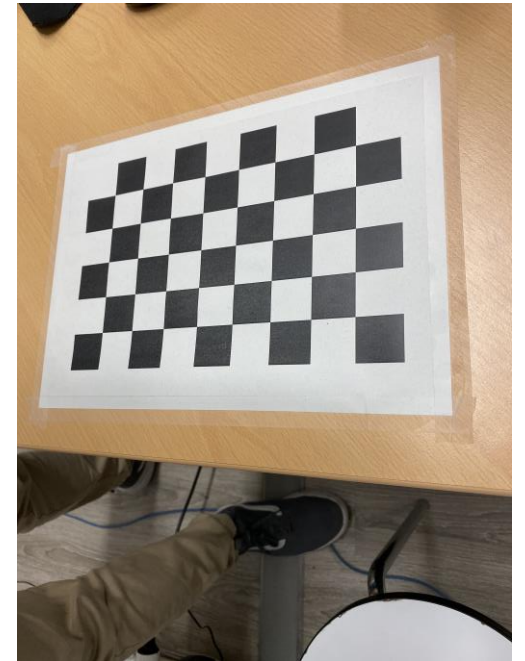
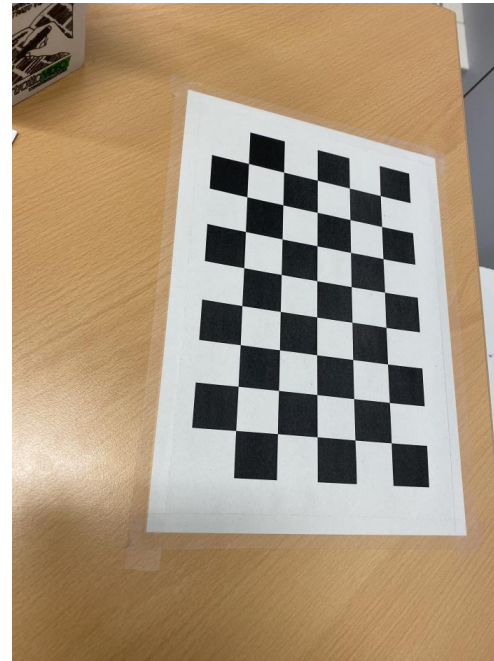
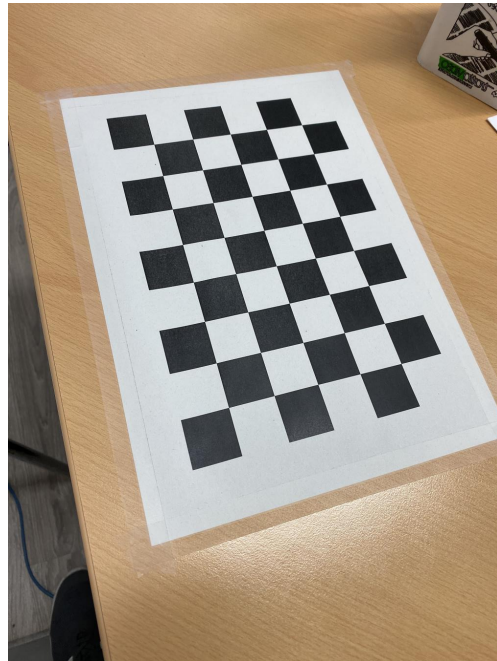
Given a chessboard pattern, we know:

- The **number of internal corners** of the pattern, usually odd along one dimension and even along the other to remove rotation ambiguities.
- The **size of the squares** that form the pattern (in mm, cm...)

Internal corners can be detected easily by standard algorithms (e.g. the Harris corner detector)



Ambiguous



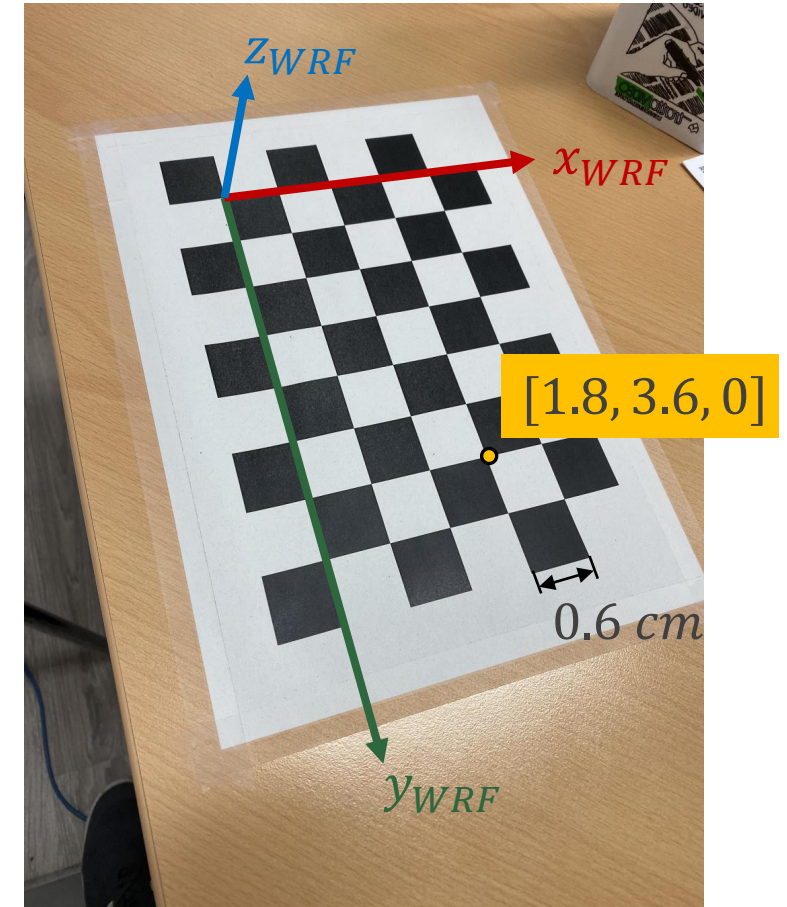
3D coordinates

The World Reference Frame can be conveniently defined.

In an unambiguous pattern, the WRF can be defined so that

- it has its origin always in the same corner (e.g., the one next to the dark square on the right of the chessboard if both dark squares are on top);
- its plane $z = 0$ is the pattern itself => the third coordinate is always 0;
- the x, y axes are aligned to the chessboard (e.g., x along the short side and y along the long one).

Given such rules and the known square side, it is possible to define 3D coordinates for all corners in an image of the pattern.

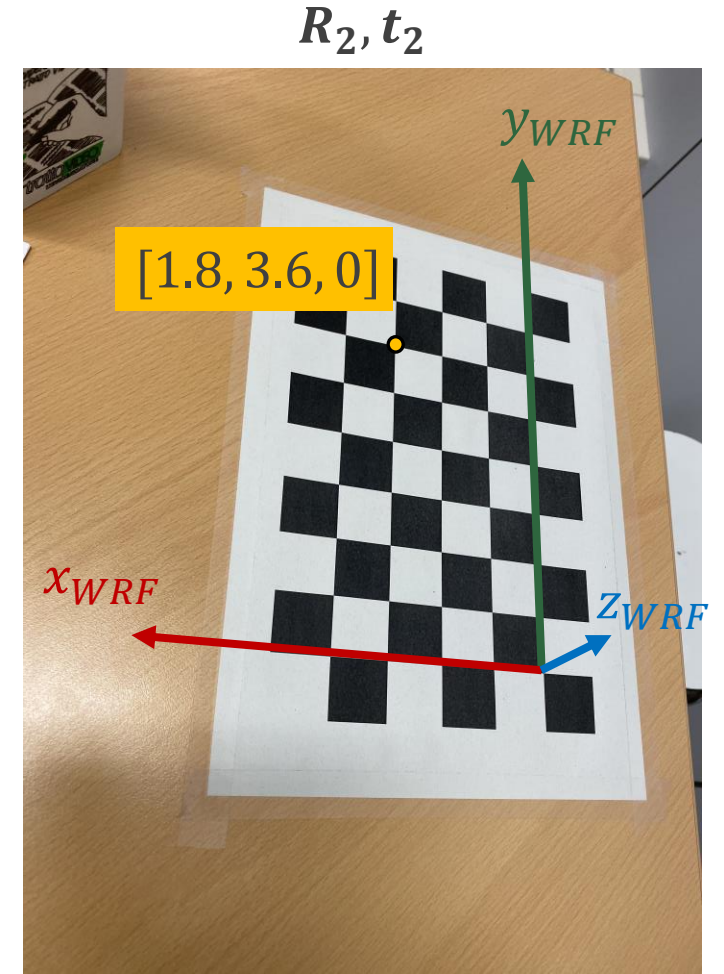
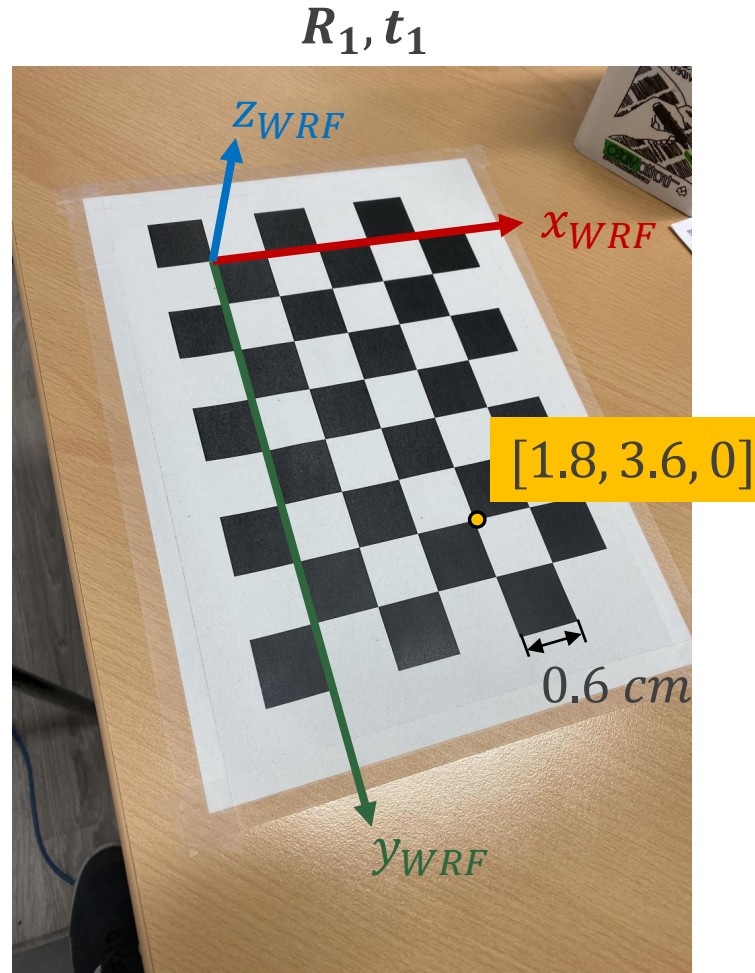


Extrinsic parameters

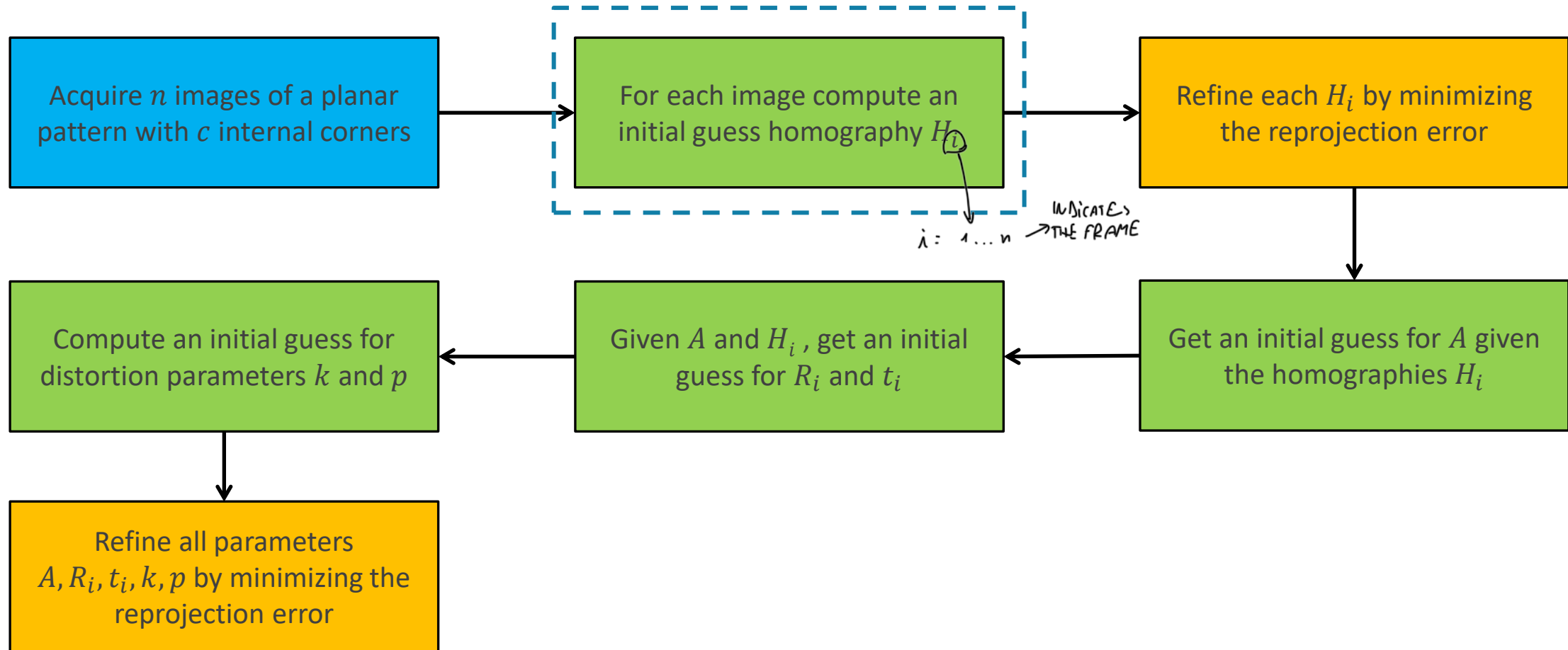
The World Reference Frame is different for each calibration image.

The $[R \ t]$ are estimated wrt the World Reference Frame attached to the target, which moves with the pattern.

Therefore, we estimate as many extrinsic matrices as the number of images used for calibration (usually 10 to 20 images of the pattern)



Zhang's Method



Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Trans. on PAMI, 2000.

P as a Homography

Due to the choice of the WRF associated with calibration images, in each of them we consider only 3D points with $z = 0$

Accordingly, the PPM **for points on the pattern** can be simplified to a 3x3 matrix:

$$k\tilde{\mathbf{m}} = k \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P}\tilde{\mathbf{M}}_W = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,4} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{H}\tilde{\mathbf{w}}$$

We look at a planar scene

Such a transformation, denoted here as \mathbf{H} , is known as **homography** and represents a general transformation between **projective planes**

\mathbf{H} can be thought of as a simplification of \mathbf{P} in case the imaged object is planar

Estimating H_i (DLT algorithm)

Given the i -th image of a pattern with c corners, we can write 3 linear equations for each corner j where:

- 3D coordinates are known due to the WRF definition
- 2D coordinates are known due to corners having been detected in the i -th image
- the unknowns are the 9 elements in H_i

$$\tilde{m}_{ij} \equiv \begin{bmatrix} u_{ij} \\ v_{ij} \\ 1 \end{bmatrix} \equiv \begin{bmatrix} p_{i,1,1} & p_{i,1,2} & p_{i,1,4} \\ p_{i,2,1} & p_{i,2,2} & p_{i,2,4} \\ p_{i,3,1} & p_{i,3,2} & p_{i,3,4} \end{bmatrix} \begin{bmatrix} x_j \\ y_j \\ 1 \end{bmatrix} \equiv H_i \tilde{w}_j \equiv \begin{bmatrix} -h_{i1}^T & - \\ -h_{i2}^T & - \\ -h_{i3}^T & - \end{bmatrix} \tilde{w}_j \equiv \begin{bmatrix} h_{i1}^T \tilde{w}_j \\ h_{i2}^T \tilde{w}_j \\ h_{i3}^T \tilde{w}_j \end{bmatrix}$$

Diagram annotations:

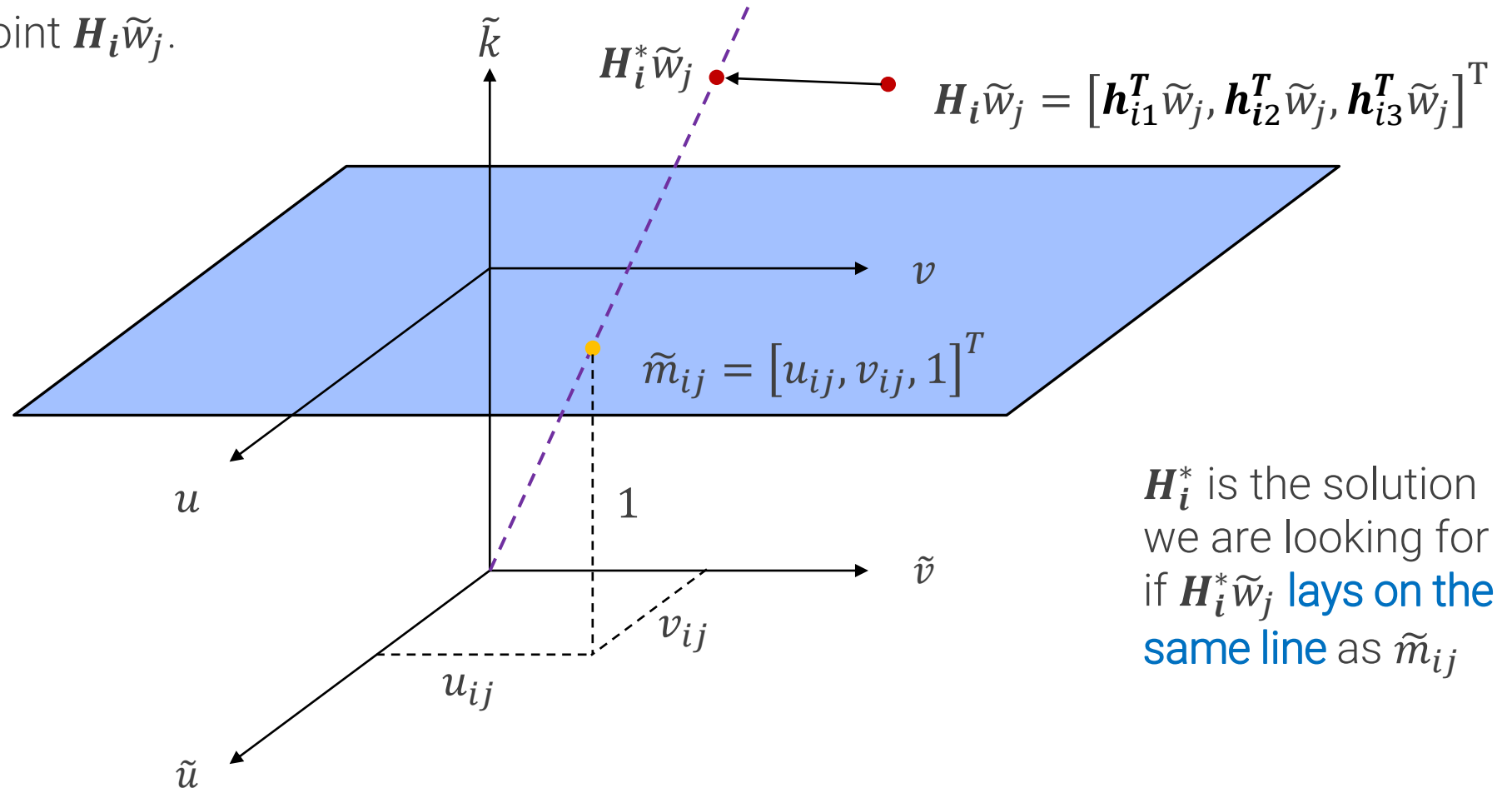
- A red box labeled "Unknowns" points to the matrix H_i .
- A blue box labeled "Coordinates of a corner in the image" points to the vector \tilde{m}_{ij} .
- A blue box labeled "Coordinates of the same corner in the WRF" points to the vector \tilde{w}_j .

Stacking $3c$ such equations for the c corners we get a system of equations, but...

How do we solve a system of equations in a projective space?

When are two 3D points equivalent in \mathbb{P}^2 ?

By deciding a value for the 9 entries of \mathbf{H}_i we move the point $\mathbf{H}_i \tilde{\mathbf{w}}_j$.



Estimating H_i (DLT algorithm)

Two points lay on the same line if their cross product is the zero vector.

Two points lay on the same line if their cross product is the zero vector.

$$\tilde{m}_{ij} \equiv H_i \tilde{w}_j \Rightarrow \tilde{m}_{ij} \times H_i \tilde{w}_j = \mathbf{0} \Rightarrow \tilde{m}_{ij} \times H_i \tilde{w}_j = \begin{bmatrix} u_{ij} \\ v_{ij} \\ 1 \end{bmatrix} \times \begin{bmatrix} h_{i1}^T \tilde{w}_j \\ h_{i2}^T \tilde{w}_j \\ h_{i3}^T \tilde{w}_j \end{bmatrix} = \begin{bmatrix} v_{ij} h_{i3}^T \tilde{w}_j - h_{i2}^T \tilde{w}_j \\ h_{i1}^T \tilde{w}_j - u_{ij} h_{i3}^T \tilde{w}_j \\ u_{ij} h_{i2}^T \tilde{w}_j - v_{ij} h_{i1}^T \tilde{w}_j \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

3x9 matrix

9x1 column vector

The 3rd line is a linear combination of the first 2, so once you obtained the first 2, the third is satisfied

Only 2 equations are linearly independent (multiply first one by $-u$ and second one by $-v$, then sum them to get the third one)

$$\begin{bmatrix} \mathbf{0}_{3 \times 1}^T & -\tilde{w}_j^T & v_{ij} \tilde{w}_j^T \\ \tilde{w}_j^T & \mathbf{0}_{3 \times 1}^T & -u_{ij} \tilde{w}_j^T \\ -v_{ij} \tilde{w}_j^T & u_{ij} \tilde{w}_j^T & \mathbf{0}_{3 \times 1}^T \end{bmatrix} \begin{bmatrix} h_{i1} \\ h_{i2} \\ h_{i3} \end{bmatrix} = \mathbf{0}_{3 \times 1}$$

Each of these vectors is 3x1

$$h_k^T \tilde{w}_j = \tilde{w}_j^T h_k$$

The dot product is commutative

$$\begin{bmatrix} \mathbf{0}_{3 \times 1}^T & -\tilde{w}_j^T & v_{ij} \tilde{w}_j^T \\ \tilde{w}_j^T & \mathbf{0}_{3 \times 1}^T & -u_{ij} \tilde{w}_j^T \end{bmatrix} \begin{bmatrix} h_{i1} \\ h_{i2} \\ h_{i3} \end{bmatrix} = \mathbf{0}_{2 \times 1}$$

2x9 matrix

Estimating H_i (DLT algorithm)

Given c corners, we can create a homogeneous, overdetermined linear system of equations

$$\begin{bmatrix} \mathbf{0}_{3 \times 1}^T & -\tilde{\mathbf{w}}_1^T & v_{i1} \tilde{\mathbf{w}}_1^T \\ \tilde{\mathbf{w}}_1^T & \mathbf{0}_{3 \times 1}^T & -u_{i1} \tilde{\mathbf{w}}_1^T \\ \vdots & \vdots & \vdots \\ \mathbf{0}_{3 \times 1}^T & -\tilde{\mathbf{w}}_c^T & v_{ic} \tilde{\mathbf{w}}_c^T \\ \tilde{\mathbf{w}}_c^T & \mathbf{0}_{3 \times 1}^T & -u_{ic} \tilde{\mathbf{w}}_c^T \end{bmatrix} \begin{bmatrix} h_{i1} \\ h_{i2} \\ h_{i3} \end{bmatrix} = \mathbf{0}_{2c \times 1} \Rightarrow L_i \mathbf{h}_i = \mathbf{0}$$

2c x 9 matrix
9 x 1 column vector

Where $c \gg 9$

To avoid the trivial solution $\mathbf{h} = \mathbf{0}$ we look for solutions with an additional constraint, e.g., $\|\mathbf{h}\| = 1$.

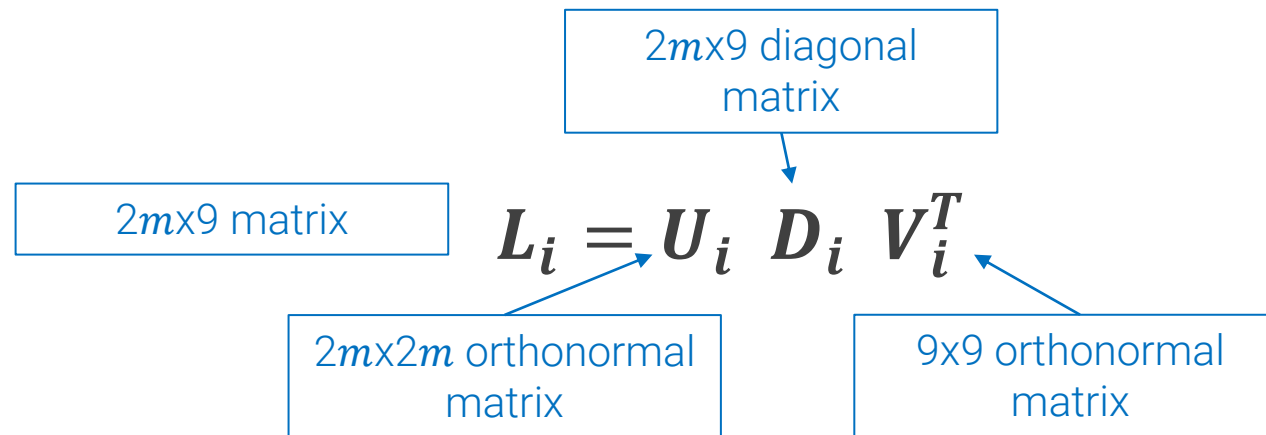
This is impossible to solve, so we relax our constraint in order to have \mathbf{h}_j
not to map exactly but to find the most close solution as possible

Singular Value Decomposition

The solution \mathbf{h}^* is found by minimizing the norm of the vector $\mathbf{L}_i \mathbf{h}_i$

$$\mathbf{h}_i^* = \underset{\mathbf{h}_i \in \mathbb{R}^9}{\operatorname{argmin}} \|\mathbf{L}_i \mathbf{h}_i\| \quad \text{s.t.} \quad \|\mathbf{h}_i\| = 1$$

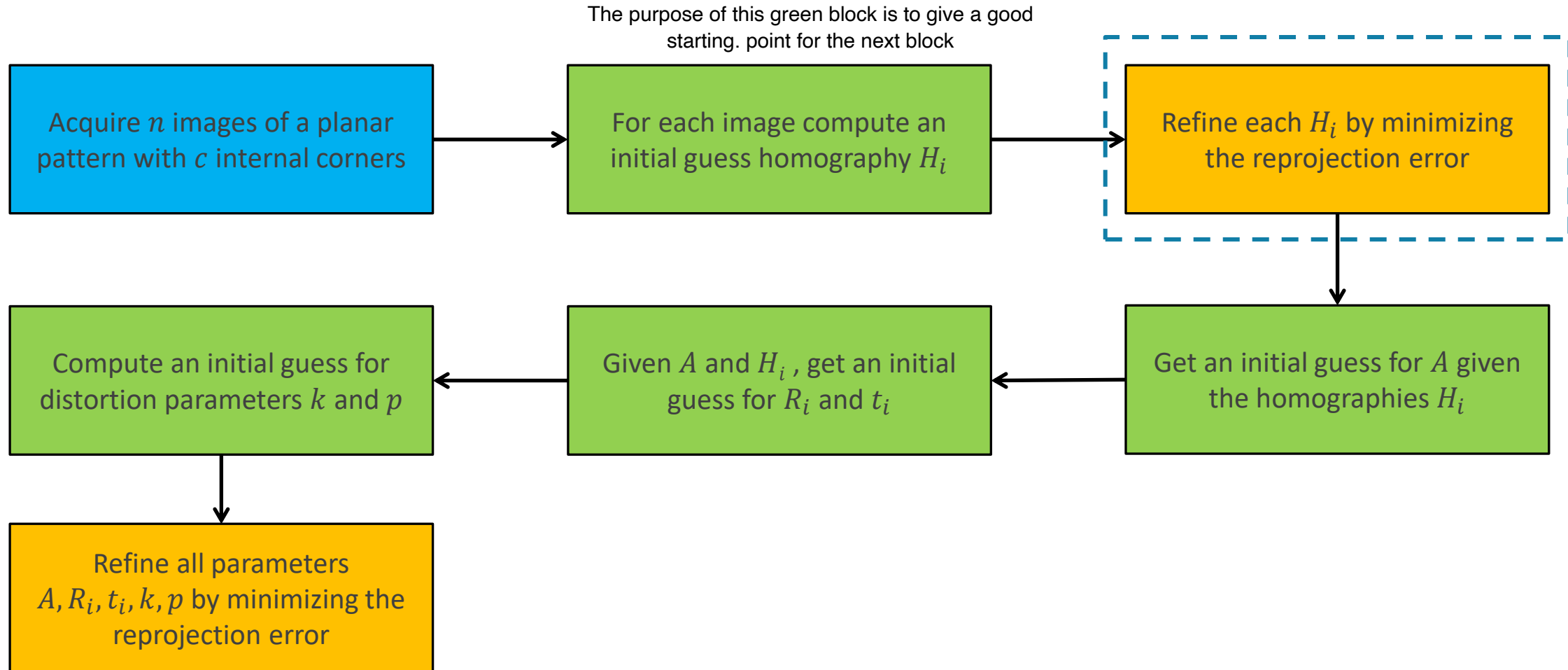
It is known from linear algebra that the solution to such problem can be found via **Singular Value Decomposition** of \mathbf{L}_i . In particular, the solution is $\mathbf{h}_i^* = \mathbf{v}_9$, i.e., the last column of \mathbf{V}_i



every matrix can be represented as:

- 1) rotation
- 2) scale axes
- 3) rotation

Zhang's Method

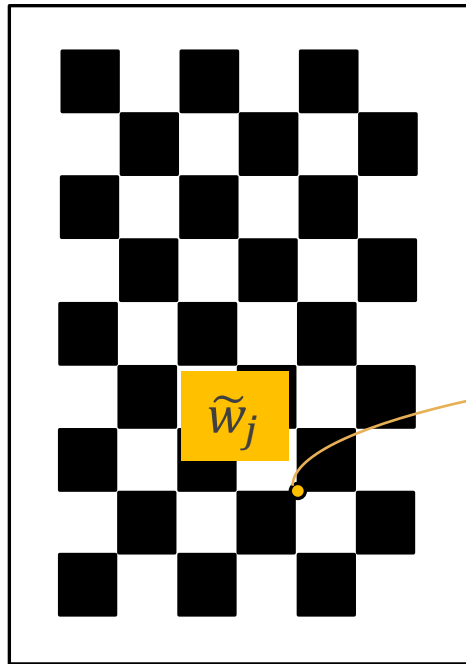


Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Trans. on PAMI, 2000.

What error should we minimize?

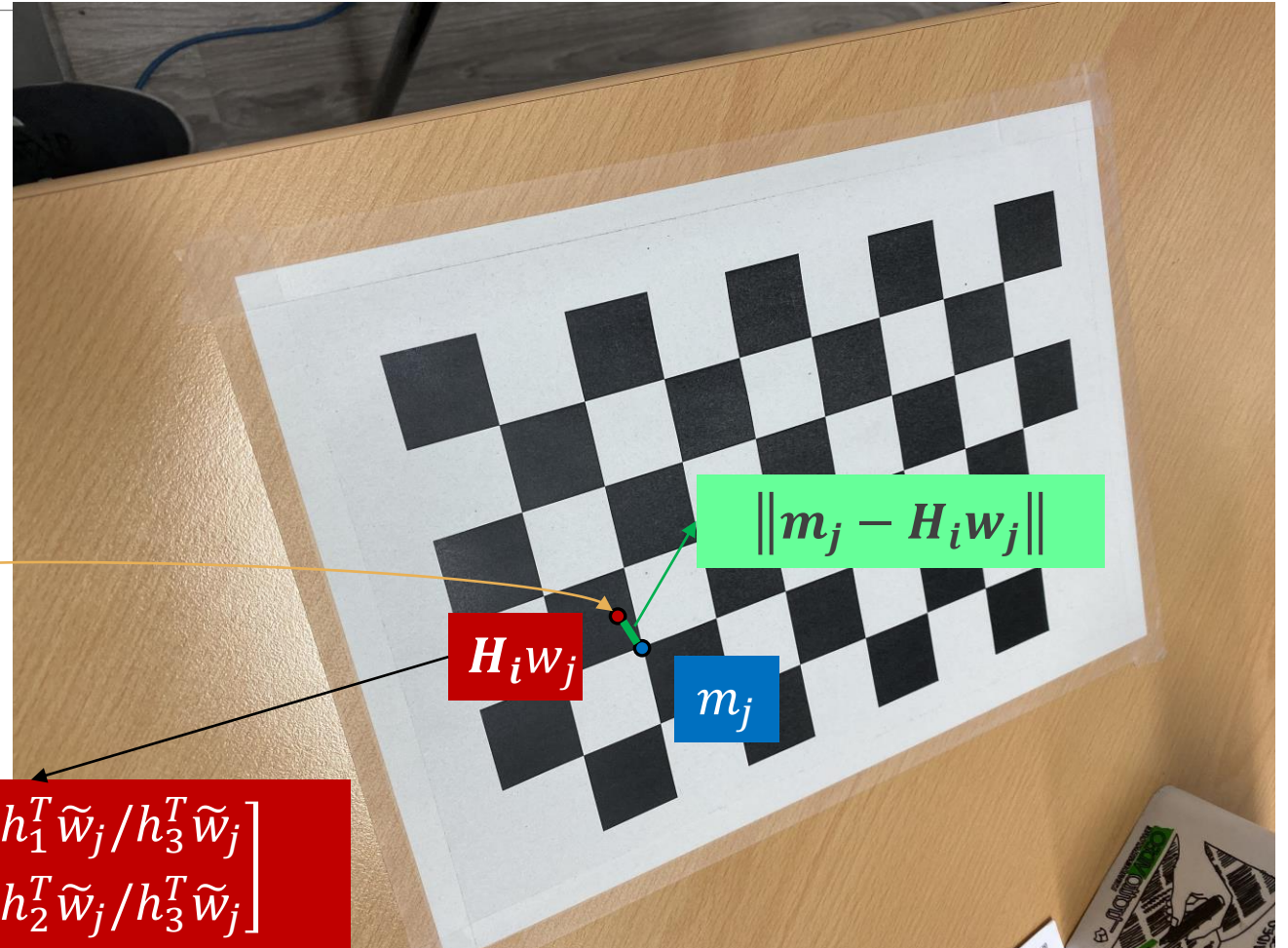
We have to minimize the distance between:

- m_j \rightarrow real position
- $H_i w_j$ \rightarrow estimation



Why don't we solve the distance in the first place?
- Because it is ...

$H_i w_j = \begin{bmatrix} h_1^T \tilde{w}_j / h_3^T \tilde{w}_j \\ h_2^T \tilde{w}_j / h_3^T \tilde{w}_j \end{bmatrix}$
is a 2x1 Euclidean vector



Non-linear refinement of H_i

Given the initial guess for \mathbf{H}_i , we can refine it by a non-linear minimization problem:

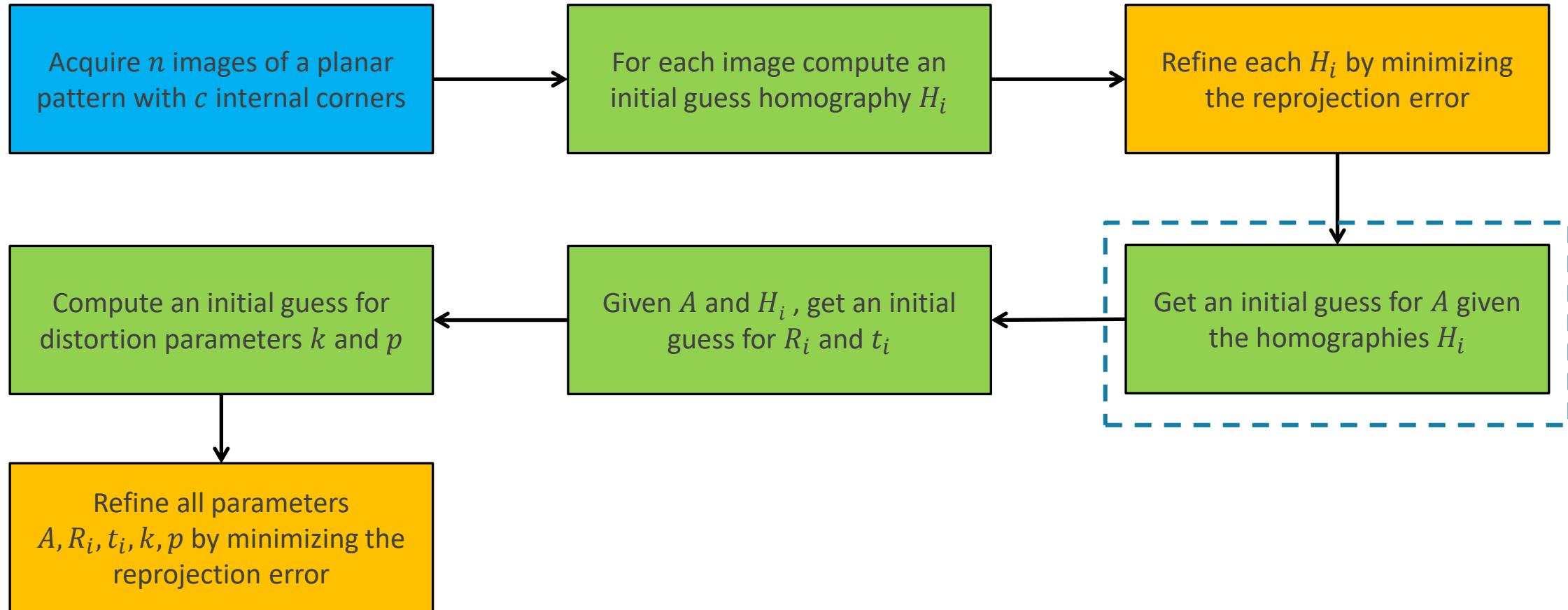
$$\mathbf{H}_i^* = \operatorname{argmin}_{\mathbf{H}_i} \sum_{j=1}^c \|\mathbf{m}_{ij} - \mathbf{H}_i \mathbf{w}_j\|^2 \quad i = 1, \dots, n$$

which can be solved for by using an [iterative algorithm](#), like the Levenberg-Marquardt algorithm.

This additional optimization step corresponds to the minimization of the [reprojection error](#) (typically referred to as [geometric error](#)) measured for each of the 3D corners of the pattern by comparing the pixel coordinates predicted by the estimated homography to the pixel coordinates of the corresponding corner extracted in the image.

The error minimized to estimate the initial guess when solving the linear system is instead referred to as [algebraic error](#) or distance. Solutions based on minimization of the algebraic error may not be aligned with our intuition, yet there exist a unique solution, which is cheap to compute. Hence, they are a good starting point for a geometric, non-linear minimization, which effectively minimize the distance we care about.

Zhang's Method



Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Trans. on PAMI, 2000.

Estimation of the intrinsic parameters

All the images acquired for calibration share the same **intrinsic parameters**

We can establish the following relations between them and the extrinsic and intrinsic parameters

$$\begin{aligned} P_i \equiv A[R_i|t_i] &= A[r_{i1} \quad r_{i2} \quad r_{i3} \quad t_i] \Rightarrow H_i = [h_{i1} \quad h_{i2} \quad h_{i3}] = [kAr_{i1} \quad kAr_{i2} \quad kAt_i] \\ &\Rightarrow k r_{i1} = A^{-1}h_{i1}, \quad k r_{i2} = A^{-1}h_{i2} \end{aligned}$$

We don't know k, r, A \rightarrow we only know h

But we know that each r is a column of R

Since the column vectors of each R_i are orthonormal, we get the following constraints

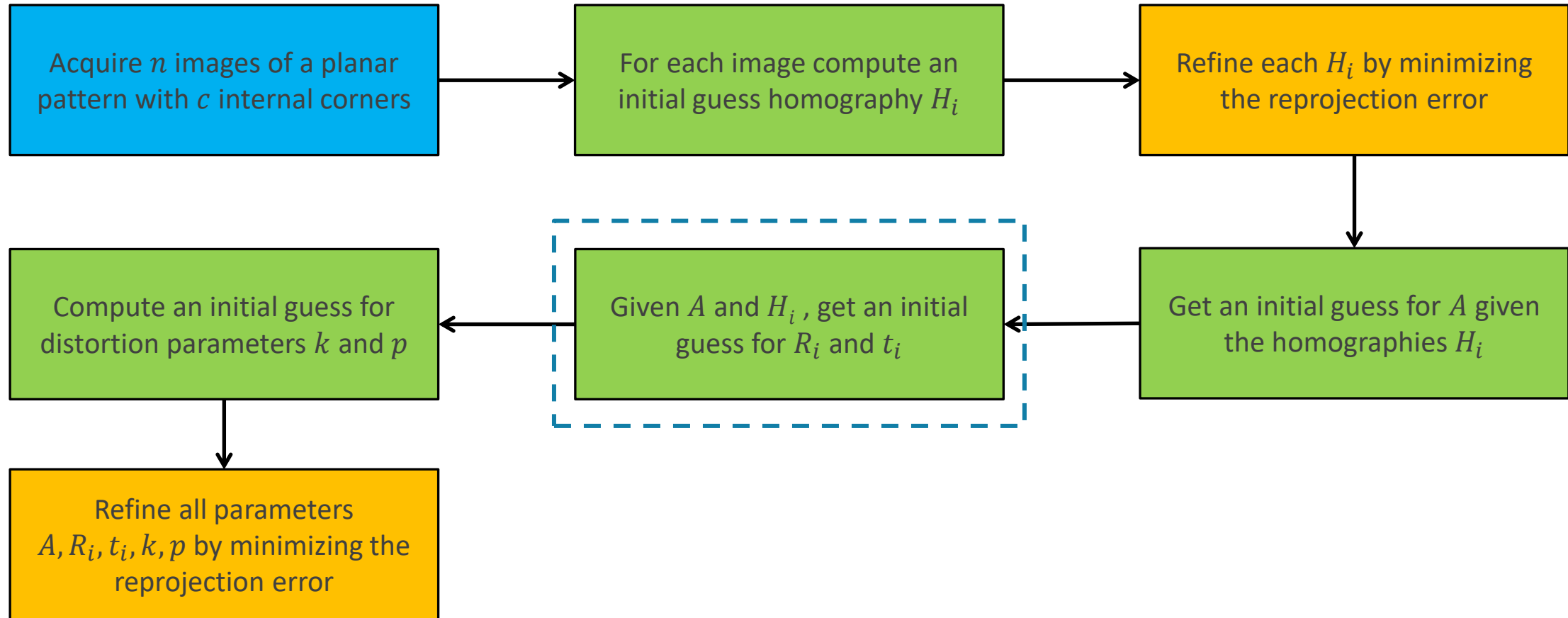
$$\text{Orthogonality } \langle r_{i1}, r_{i2} \rangle = 0 \Rightarrow \langle A^{-1}h_{i1}, A^{-1}h_{i2} \rangle = 0 \Rightarrow h_{i1}^T A^{-T} A^{-1} h_{i2} = 0$$

$$\text{Unit length } \langle r_{i1}, r_{i1} \rangle = \langle r_{i2}, r_{i2} \rangle \Rightarrow \langle A^{-1}h_{i1}, A^{-1}h_{i1} \rangle = \langle A^{-1}h_{i2}, A^{-1}h_{i2} \rangle \Rightarrow h_{i1}^T A^{-T} A^{-1} h_{i1} = h_{i2}^T A^{-T} A^{-1} h_{i2}$$

By stacking these two constraints for each image, we get a homogeneous system of equations which can be solved again by SVD if $n \geq 3$ images are collected (6 unknowns since $A^{-T}A^{-1}$ is symmetric).

We need more than 3 images also for this \rightarrow

Zhang's Method



Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Trans. on PAMI, 2000.

Estimation of the extrinsic parameters

Once \mathbf{A} has been estimated, it is possible to compute \mathbf{R}_i and \mathbf{t}_i (for each image) given \mathbf{A} and the previously computed homography \mathbf{H}_i :

$$\mathbf{H}_i = [\mathbf{h}_{i1} \quad \mathbf{h}_{i2} \quad \mathbf{h}_{i3}] = [k\mathbf{A}\mathbf{r}_{i1} \quad k\mathbf{A}\mathbf{r}_{i2} \quad k\mathbf{A}\mathbf{t}_i] \Rightarrow \mathbf{r}_{i1} = \frac{1}{k}\mathbf{A}^{-1}\mathbf{h}_{i1}$$

As \mathbf{r}_{i1} is a unit vector, the normalization constant can be computed as $k = \|\mathbf{A}^{-1}\mathbf{h}_{i1}\|$

Then, the same constant can be used to compute

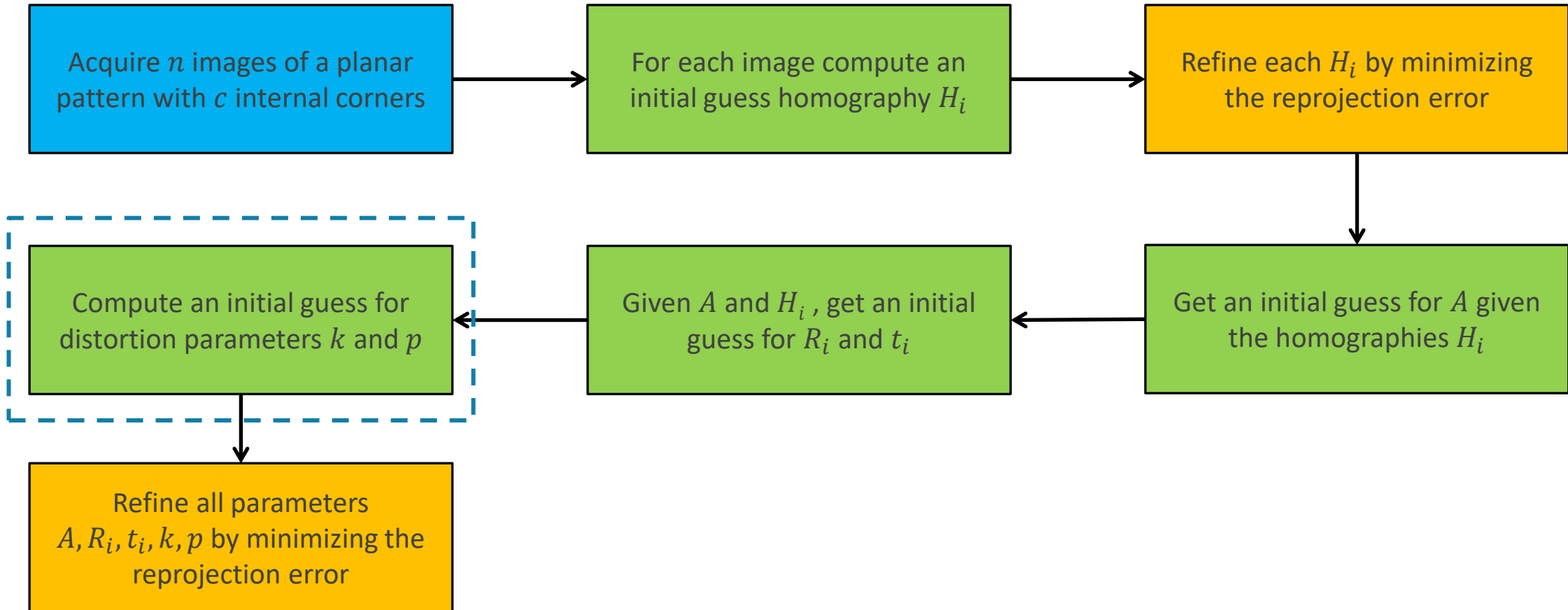
$$\mathbf{r}_{i2} = \frac{1}{k}\mathbf{A}^{-1}\mathbf{h}_{i2} \quad \text{and} \quad \mathbf{t}_i = \frac{1}{k}\mathbf{A}^{-1}\mathbf{h}_{i3}$$

Finally, enforcing again orthonormality of \mathbf{R}_i , $\mathbf{r}_{i3} = \mathbf{r}_{i1} \times \mathbf{r}_{i2}$

Yet, the resulting matrix \mathbf{R}_i will not be exactly orthonormal since \mathbf{r}_{i1} and \mathbf{r}_{i2} are not necessarily orthogonal and \mathbf{r}_{i2} does not necessarily have unit length since k was computed for \mathbf{r}_{i1} .

However, SVD of \mathbf{R}_i allows to find the closest orthonormal matrix to it by substituting \mathbf{D} with \mathbf{I} .

Zhang's Method



Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Trans. on PAMI, 2000.

Lens distortion coefficients

So far, we have neglected lens distortion and calibrated a pure pinhole model. The coordinates predicted by the homographies starting from points in the WRFs correspond to the ideal (undistorted) pixel coordinates of the chessboard corners m_{undist} . The measured coordinates of the corners in the images are the real (distorted) coordinates m .

Original Zhang's method deploys such information to estimate coefficients k_1, k_2 of the radial distortion function:

$$\begin{bmatrix} x \\ y \end{bmatrix} = L(r) \begin{bmatrix} x_{undist} \\ y_{undist} \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4) \begin{bmatrix} x_{undist} \\ y_{undist} \end{bmatrix}$$

OpenCV uses a different method for estimating the distortion parameters:

- 3 coefficients for radial distortion (k_1, k_2, k_3)
- 2 coefficients for tangential distortion (p_1, p_2)

Metric image coordinates

Recall: lens distortion takes place **before** we change metric image coordinates to pixel coordinates. But we measure and predict pixel coordinates.

We can transform back pixel coordinates $\begin{bmatrix} u \\ v \end{bmatrix}$ to metric image coordinates $\begin{bmatrix} x \\ y \end{bmatrix}$ thanks to the estimated intrinsic matrix \mathbf{A}

$$\begin{bmatrix} ku \\ kv \\ k \end{bmatrix} \equiv \mathbf{A} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} ku \\ kv \\ k \end{bmatrix} \equiv \begin{bmatrix} f_u x + u_0 \\ f_v y + v_0 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{u-u_0}{f_u} \\ \frac{v-v_0}{f_v} \end{bmatrix}$$


The same transformation holds between u_{undist} , v_{undist} and x_{undist} , y_{undist}

Then, the distortion equation in pixel coordinates become

$$\begin{bmatrix} \frac{u-u_0}{f_u} \\ \frac{v-v_0}{f_v} \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4) \begin{bmatrix} \frac{u_{undist}-u_0}{f_u} \\ \frac{v_{undist}-v_0}{f_v} \end{bmatrix} \Rightarrow \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4) \begin{bmatrix} u_{undist} - u_0 \\ v_{undist} - v_0 \end{bmatrix}$$

Multiplication
by
f_u
and
f_v

Lens distortion coefficients

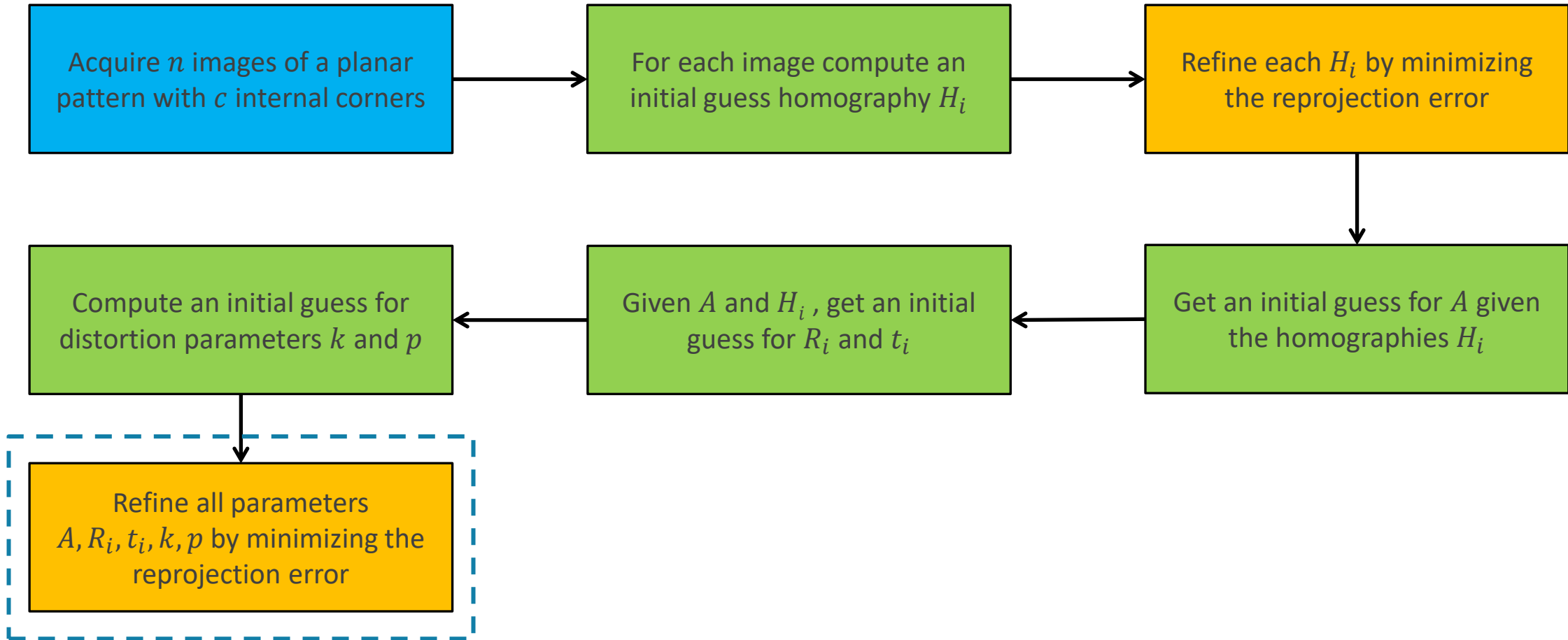
$$\begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4) \begin{bmatrix} u_{undist} - u_0 \\ v_{undist} - v_0 \end{bmatrix} \Rightarrow \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} - \begin{bmatrix} u_{undist} - u_0 \\ v_{undist} - v_0 \end{bmatrix} = (k_1 r^2 + k_2 r^4) \begin{bmatrix} u_{undist} - u_0 \\ v_{undist} - v_0 \end{bmatrix}$$


$$\begin{bmatrix} u - u_{undist} \\ v - v_{undist} \end{bmatrix} = \begin{bmatrix} (u_{undist} - u_0)r^2 & (u_{undist} - u_0)r^4 \\ (v_{undist} - v_0)r^2 & (v_{undist} - v_0)r^4 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix}$$

We get a **linear, non-homogeneous** system of linear equations $\mathbf{D}\mathbf{k} = \mathbf{d}$ in the unknowns $\mathbf{k} = [k_1 \ k_2]^T$. With c corners in n images we get $2nc$ equations in 2 unknowns, which can be solved in a least square sense, i.e., minimizing $\|\mathbf{D}\mathbf{k} - \mathbf{d}\|_2$, by computing the pseudo-inverse matrix \mathbf{D}^\dagger as

$$\mathbf{k}^* = \min_{\mathbf{k}} \|\mathbf{D}\mathbf{k} - \mathbf{d}\|_2 = \mathbf{D}^\dagger \mathbf{d} = (\mathbf{D}^T \mathbf{D})^{-1} \mathbf{D}^T \mathbf{d}$$

Zhang's Method



Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Trans. on PAMI, 2000.

Refinement by non-linear optimization

Final non-linear refinement of the estimated parameters. As for homographies, the procedure highlighted so far seeks to minimize an **algebraic error**, without any real physical meaning.

A more accurate solution can instead be found by a so called Maximum Likelihood Estimate (MLE) aimed at minimization of the geometric (i.e. reprojection) error.

We use all the values estimated so far as **initial guesses**.

Under the hypothesis of i.i.d. (independent identically distributed) noise, the MLE for our models is obtained by minimization of the error

$$\mathbf{A}^*, \mathbf{k}^*, \mathbf{R}_i^*, \mathbf{t}_i^* = \underset{\mathbf{A}, \mathbf{k}, \mathbf{R}_i, \mathbf{t}_i}{\operatorname{argmin}} \sum_{i=1}^n \sum_{j=1}^c \|\tilde{\mathbf{m}}_{ij} - \hat{\mathbf{m}}(\mathbf{A}, \mathbf{k}, \mathbf{R}_i, \mathbf{t}_i, \tilde{\mathbf{w}}_j)\|^2$$

with respect to all the unknown camera parameters, which can be solved again by using an **iterative algorithm**, like the Levenberg-Marquardt algorithm.

Compensate lens distortion

After calibration, we have a precise mathematical model that maps points in the 3D world to points in the image plane. Yet, lens distortion makes the system non-linear and therefore cumbersome to use.

Hence, it is common, once a camera has been calibrated, to **warp** the images it takes so to simulate a camera without lens distortion, whose camera formation model is linear, i.e. it is the estimated PPM.



undistort



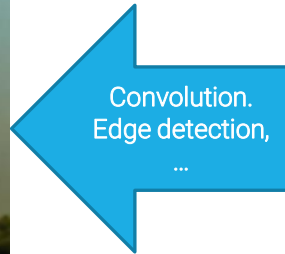
<https://docs.nvidia.com/isaac/archive/2020.2/packages/perception/doc/warp.html>

Image Warping \neq Image filtering

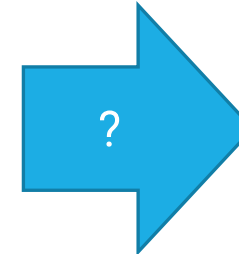
Warping refers to transformations of the **spatial domain** of images, while **filtering** refers to changes in the **RGB values** of images



Filtered (Gaussian blur)



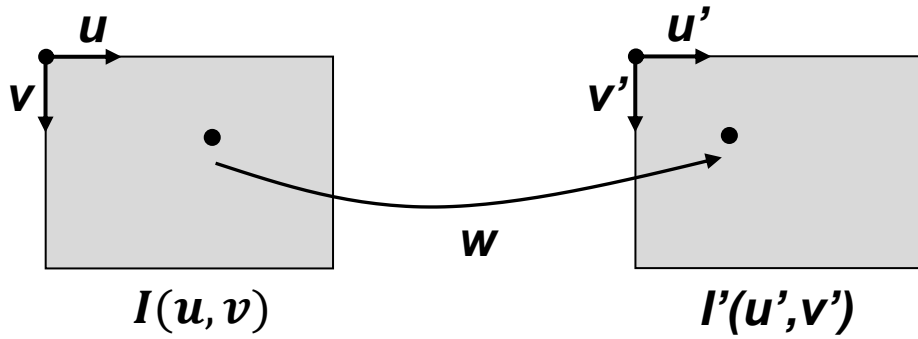
Original



Warped (Rotation & scale)

Image Warping

If we have a function that computes point in image I' starting from point in image I , we can copy the value

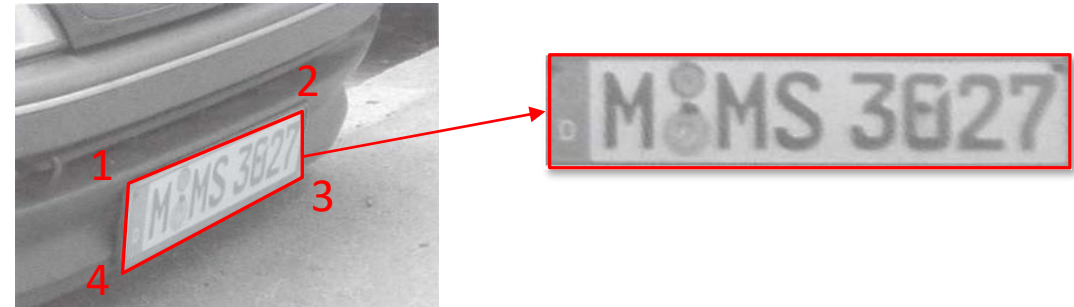


$$\begin{cases} u' = w_u(u, v) \\ v' = w_v(u, v) \end{cases}$$

$$I'(w_u(u, v), w_v(u, v)) = I(u, v)$$

Can be just a rotation $\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$

Or it can be a full homography $k \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{32} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$



Forward mapping

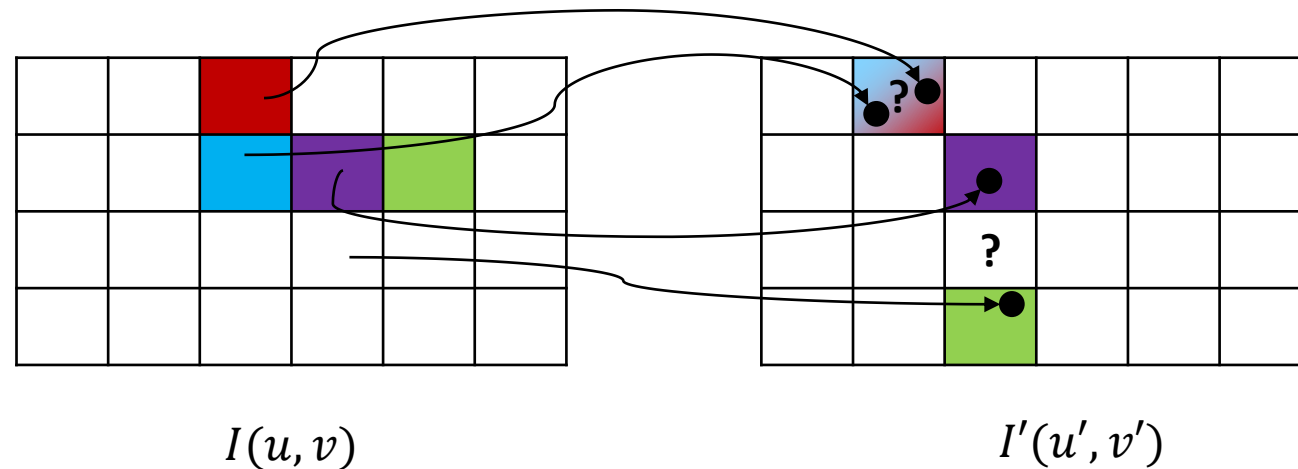
If we start from the input image coordinates, after applying the warping function in general we get **continuous coordinates in the output image**, not discrete ones. This is called a **forward mapping**.

Possible choices to make coordinates integer numbers: truncate, **nearest neighbor** (i.e. rounding), ... Regardless of the discretization function, due to rounding

- **more than one pixel can go to one position (folds)** Rounding is not biunivocal so we can have different points rounded to the same indexes
- **some pixels of the destination image may not be hit (holes)** For the same reason some integer numbers may never be hit by rounding

$$\begin{cases} u' = w_u(u, v) \\ v' = w_v(u, v) \end{cases}$$

I will obtain not integer coordinates \rightarrow rounding



Backward mapping

We can avoid these problems if we compute input coordinates corresponding to **each pair of integer coordinates in the output image**, by using the **inverse mapping** w^{-1} . This strategy is referred to as **backward mapping**.

Yet, the input coordinates are still continuous values. Which discretization strategy are used?

-Truncate

-Nearest Neighbour

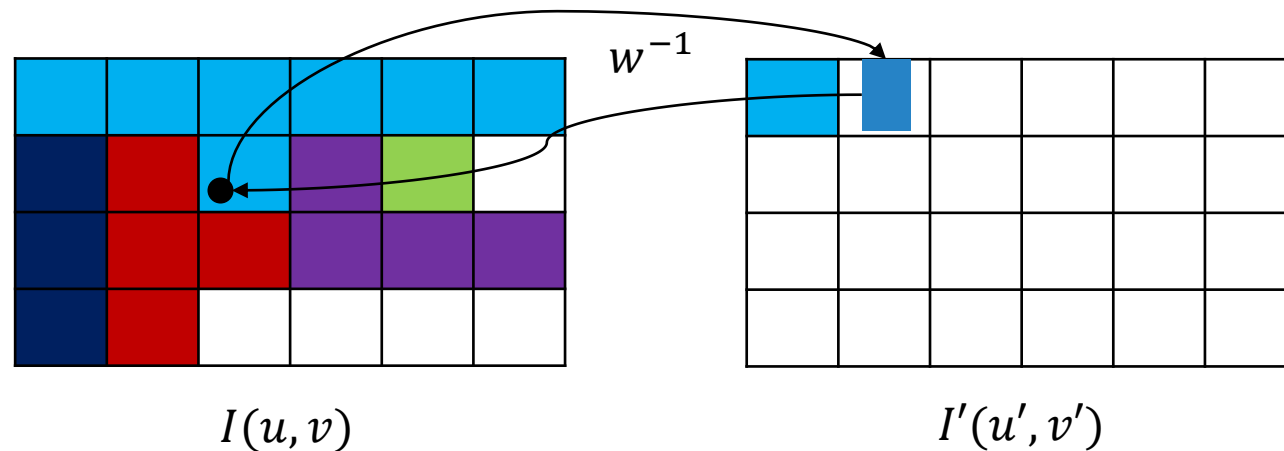
-Interpolate between the 4 closest point (bilinear, bicubic, etc...)

So we start from the destination image instead of the source one.

We choose the empty cell where to put a colour in the I' , we look at the source and we obtain the non integer coordinates,

...

$$\begin{cases} u = w_u^{-1}(u', v') \\ v = w_v^{-1}(u', v') \end{cases}$$

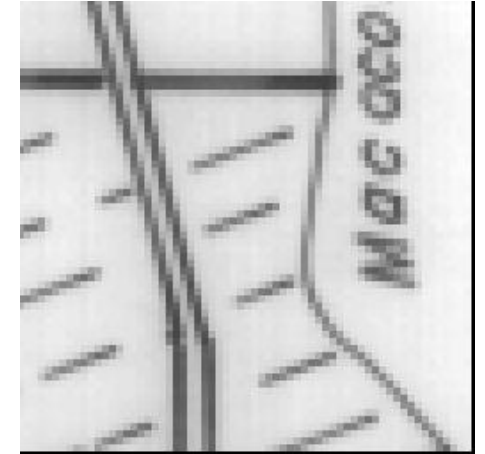


Bilinear Interpolation



Input

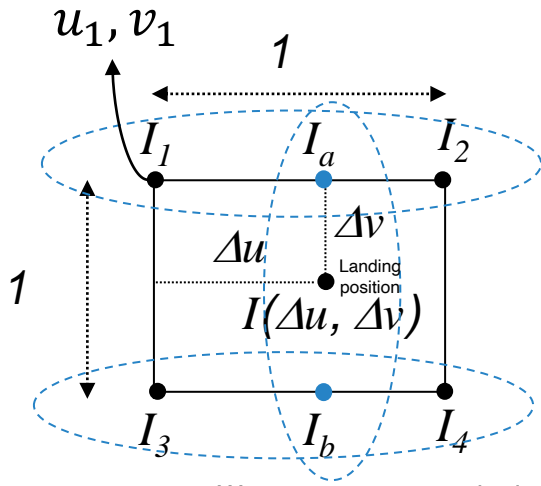
Warp (Zoom) by nearest neighbor



Warp (Zoom) by Bilinear Interpolation

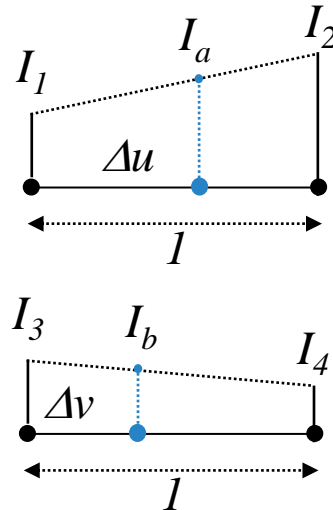


We assume I_2 brighter than I_1 and I_3 brighter than I_4



We want to compute the interpolation between I_a and I_b

The closer a point to a pixel the smaller the other weights become



$$\Delta u = u - u_1$$

$$\Delta v = v - v_1$$

$$\frac{I_a - I_1}{\Delta u} = I_2 - I_1$$

$$I_a = (I_2 - I_1)\Delta u + I_1 \text{ horizontal interpolation}$$

$$I_b = (I_4 - I_3)\Delta u + I_3 \text{ horizontal interpolation}$$

$$I(\Delta u, \Delta v) = (I_b - I_a)\Delta v + I_a \text{ vertical interpolation}$$

$$I(\Delta u, \Delta v) = ((I_4 - I_3)\Delta u + I_3 - ((I_2 - I_1)\Delta u + I_1))\Delta v + ((I_2 - I_1)\Delta u + I_1)$$

Linear combination

$$I'(u', v') = (1 - \Delta u)(1 - \Delta v)I_1 + \Delta u(1 - \Delta v)I_2 + (1 - \Delta u)\Delta vI_3 + \Delta u\Delta vI_4$$

Edges are sharper and more human readable

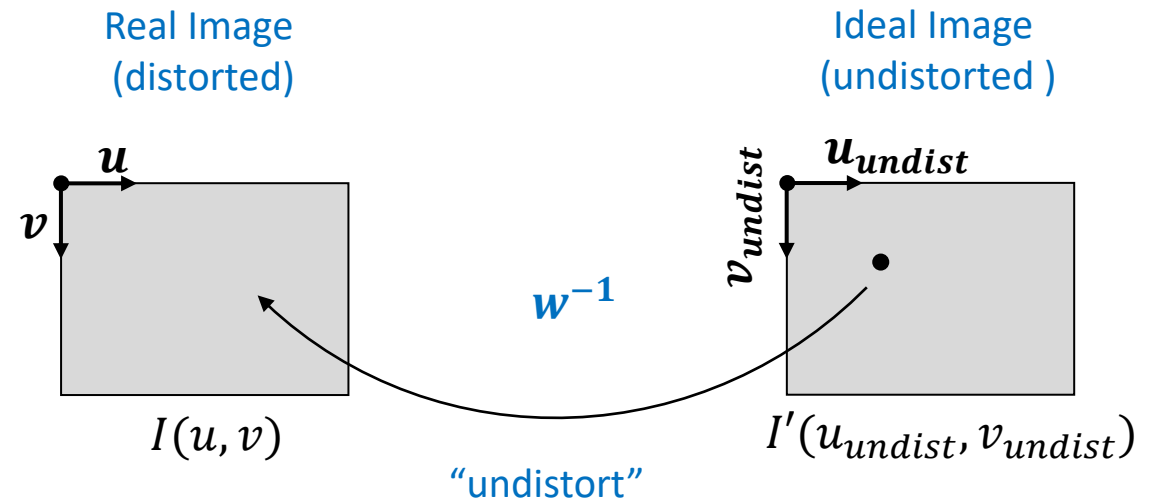
«Undistort» warping

Once the lens distortion parameters have been computed by camera calibration, the image can be corrected by a backward warp from the undistorted to the distorted image based on the adopted lens distortion model. **For this images, the image formation model is linear, i.e. the PPM.**

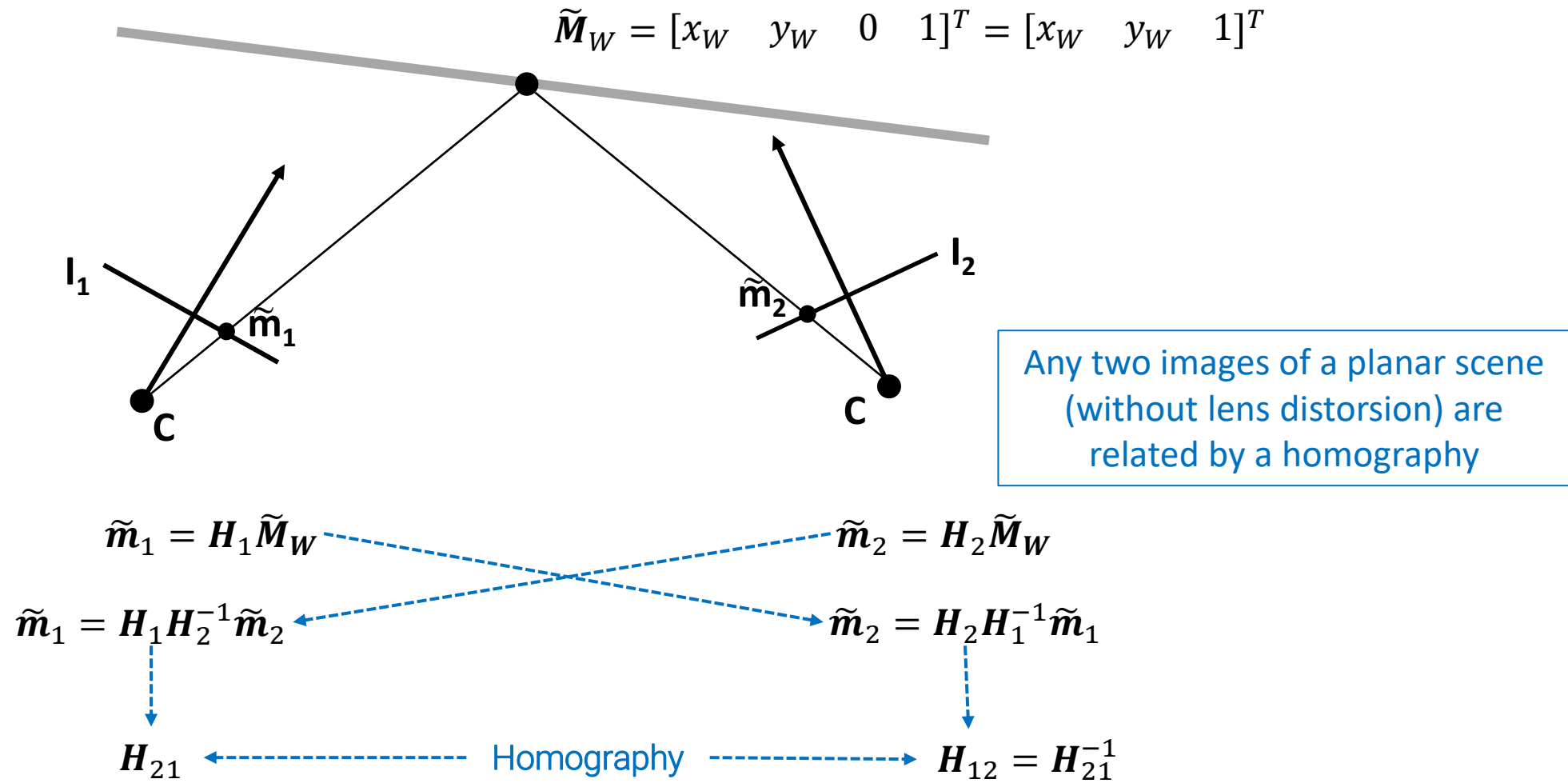
$$\forall(u_{undist}, v_{undist}): I'(u_{undist}, v_{undist}) = I(w_u^{-1}(u_{undist}, v_{undist}), w_v^{-1}(u_{undist}, v_{undist}))$$

$$w = \begin{cases} u = u_{undist} + (k_1 r^2 + k_2 r^4)(u_{undist} - u_0) \\ v = v_{undist} + (k_1 r^2 + k_2 r^4)(v_{undist} - v_0) \end{cases}$$

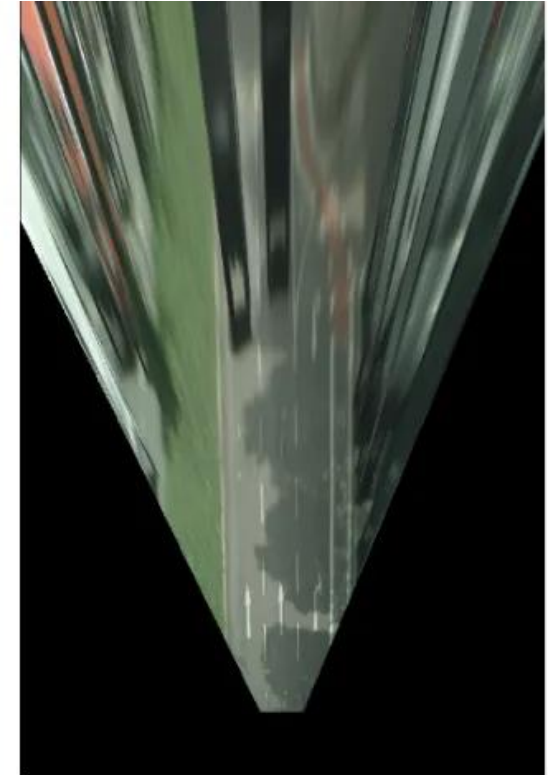
Zhang's
Radial distortion



Warping with homographies

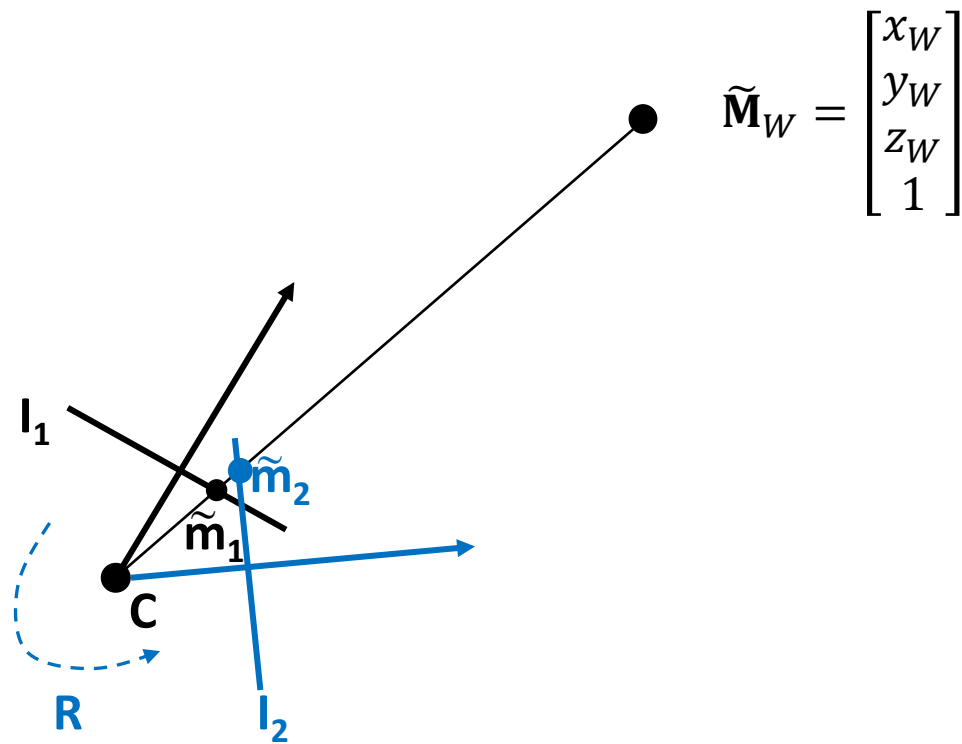


Example: Inverse Perspective Mapping



<https://towardsdatascience.com/a-hands-on-application-of-homography-ipm-18d9e47c152f>

Warping with homographies



Any two images taken by a camera rotating about its optical center are related by a homography (if lens distortion has been removed)

Rotation is identity and translation is 0

$$\tilde{m}_1 = A [I | 0] \tilde{M}_W = A M_W$$

We take the same camera and rotate it

$$\tilde{m}_2 = A [R | 0] \tilde{M}_W = A R M_W$$

Thanks to M_W

$$\tilde{m}_1 = A R^{-1} A^{-1} \tilde{m}_2$$

$$\tilde{m}_2 = A R A^{-1} \tilde{m}_1$$

Homography

$$H_{21} \leftarrow H_{12} = H_{21}^{-1}$$

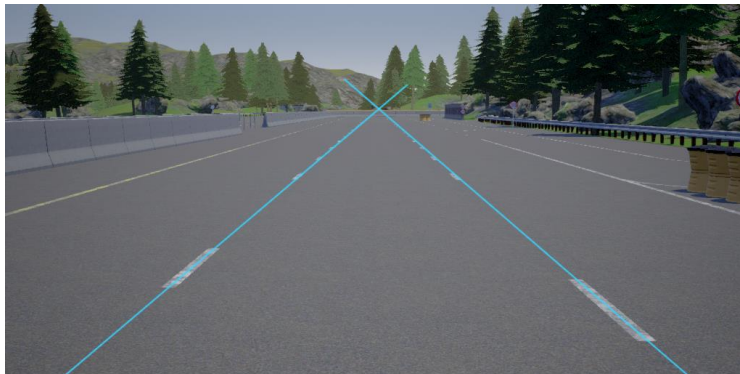
We can exaggerate it (frontal view and upper view) but it would make the image invalid due to the too big rotation

Example: compensate pitch or yaw

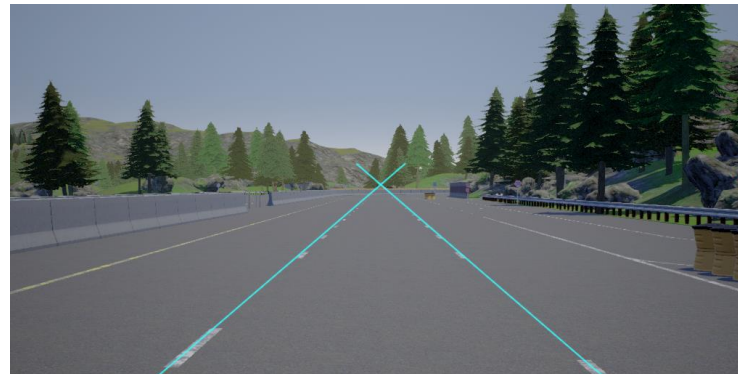
Autonomous driving cameras should be ideally mounted with the optical axis parallel to the road plane and aligned with the direction of motion. It is however difficult to obtain perfect alignment by mechanical mounting only.

With a calibrated camera, however, it is possible to **compensate pitch** (rotation around the x axis) and **yaw** (rotation around the y axis) **by estimating the vanishing point of the lane lines when the vehicle is travelling straight in a lane.**

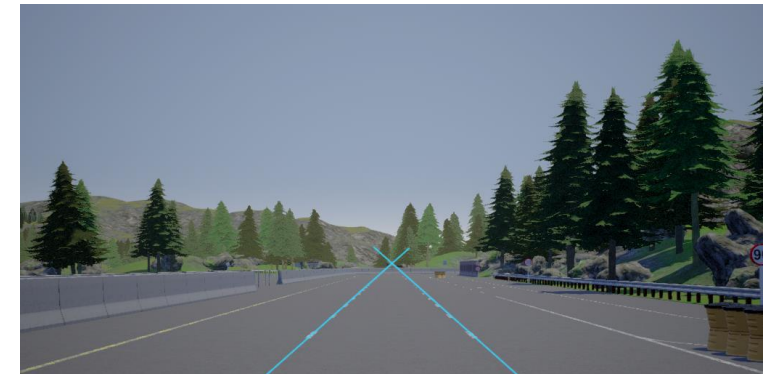
Pitch = -5°



Pitch = 0°



Pitch = $+5^\circ$



<https://thomasfermi.github.io/Algorithms-for-Automated-Driving/CameraCalibration/VanishingPointCameraCalibration.html>

Example: compensate pitch or yaw

When the vehicle is driving straight with respect to the lines, their orientation in a world reference frame attached to the vehicle is parallel to the z axis, i.e. it is $[0 \ 0 \ 1]^T$ and their point at infinity in \mathbb{P}^3 is $[0 \ 0 \ 1 \ 0]^T$.

The vanishing point is then $m_\infty \equiv A[R|0] \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \equiv Ar_3 \equiv A \begin{bmatrix} 0 \\ \sin \beta \\ \cos \beta \end{bmatrix}$ since a rotation around the x axis of an angle β has expression $R_{pitch} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \beta & \sin \beta \\ 0 & -\sin \beta & \cos \beta \end{bmatrix}$

Unknown

Then, if we estimate the coordinates of the vanishing point in the image, we can compute $r_3 = \frac{A^{-1}m_\infty}{\|A^{-1}m_\infty\|_2}$ and from it R_{pitch} and finally the warping homography from 0 pitch to input image as $AR_{pitch}A^{-1}$

With the same procedure it is possible to correct simultaneously yaw and pitch.